Tabling for infinite probability computation

Taisuke Sato[1] and Philipp Meyer[2]

1 **Tokyo Institute of Technology**
2-12-1 Ookayama, Meguro, Tokyo, Japan
sato@mi.cs.titech.ac.jp
2 **Technische Universität München**
Arcisstrasse 21, 80333 München, Germany
meyerphi@in.tum.de

Abstract

Tabling in logic programming has been used to eliminate redundant computation and also to stop infinite loop. In this paper we add the third usage of tabling, i.e. to make infinite computation possible for probabilistic logic programs. Using PRISM, a logic-based probabilistic modeling language with a tabling mechanism, we generalize prefix probability computation for PCFGs to probabilistic logic programs. Given a top-goal, we search for all SLD proofs by tabled search regardless of whether they contain loop or not. We then convert them to a set of linear probability equations and solve them by matrix operation. The solution gives us the probability of the top-goal, which, in nature, is an infinite sum of probabilities. Our generalized approach to prefix probability computation through tabling opens a way to logic-based probabilistic modeling of cyclic dependencies.

1998 ACM Subject Classification D.3.3 Language Constructs and Features

Keywords and phrases probability, tabling, PRISM

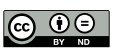
Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.348

1 Introduction

Combining logic and probability in a logic programming language provides us with a powerful modeling tool for machine learning. The resulting language allows us to build complex yet comprehensible probabilistic models in a declarative way. PRISM [12, 13, 14] is one of the earliest attempts to develop such a language. It covers a large class of known models including BNs (Bayesian networks), HMMs (hidden Markov models) and PCFGs (probabilistic context free grammars) and computes probabilities with the same time complexity as their standard algorithms[1], as well as unexplored models such as probabilistic graph grammars [11].

The efficiency of probability computation in PRISM is attributed to tabling [16, 17, 10, 20, 19][2] that eliminates redundant computation. Given a top goal G, we search for all SLD proofs of G by tabled search and translating them to a set of propositional formulas with a graph structure called an *explanation graph* for G [13]. By applying dynamic programming to the explanation graph which is acyclic and partially ordered we can efficiently compute the probability of G in proportion to the size of the graph. The use of tabling also gives us another advantage over non-tabled computation: it stops infinite loop by detecting recurrence

[1] For example, the junction tree algorithm for BNs, the forward-backward algorithm for HMMs and the inside-outside algorithm for PCFGs.
[2] Tabling is also employed in other probabilistic logic programming languages such as ProbLog [5] and PITA [9].



Technical Communications of the 28th International Conference on Logic Programming (ICLP'12).
Editors: A. Dovier and V. Santos Costa; pp. 348–358

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

patterns of goals. Tabled logic programs thus allow us to directly use left recursive rules in CFGs without the need of converting them to right recursive ones.

In this paper we pursue yet the third advantage of tabling that has gone unnoticed in the non-probabilistic setting. We apply tabling to compute an infinite sum of probabilities that typically appears in the context of prefix probability computation in PCFGs. PCFGs (probabilistic context free grammars) are a probabilistic extension of CFGs in which CFG rules have probabilities and the probability of a sentence is computed as a sum-product of probabilities assigned to the rules used to derive the sentence [1, 4]. A prefix w is an initial substring of a sentence. Then the probability of the prefix w given by a PCFG is a sum of probabilities of infinitely many complete sentences of the form wv for some string v [3, 15, 7]. Prefix probabilities are useful in speech recognition as discussed in [3]. We generalize this prefix probability computation that originated in PCFGs to probability computation on *cyclic explanation graph*s which are generated by tabled search in PRISM. We emphasize that this approach, i.e. probability computation via cyclic explanation graphs, makes it possible to model probabilistic cyclic dependencies abundant in real life from economics to biological systems by probabilistic logic programs.

Technically the generation of cyclic explanation graphs is not difficult in PRISM. Just setting appropriately a certain PRISM flag that controls tabled search is enough. However computing probabilities from such graphs is difficult in general except for the case of *linear cyclic explanation graph*s that can be turned into a set of linear probability equations straightforwardly solvable by matrix operation. So the real problem is to guarantee the linearity of cyclic explanation graphs. We specifically examine a PRISM program for prefix probability computation in PCFGs and prove that the program always generates linear cyclic explanation graphs. We also prove that the probability equations obtained from the linear cyclic explanation graphs are solvable by matrix operation under some mild assumptions on PCFGs.

2 Probability computation in PRISM

2.1 Tabling and explanation graphs

PRISM is a probabilistic extension of Prolog with built-in predicates for machine learning tasks such as parameter learning and Bayesian inference [13, 14]. Theoretically a PRISM program *DB* is a union of a set of definite clauses and a set of probabilistic atoms of the form `msw(`id`,`v`)` that simulate dice throwing[3]. It defines a probability measure (an infinite joint distribution) $P_{DB}(\cdot)$ over possible Herbrand interpretations from which the probability of an arbitrary closed formula is calculated. Practically however PRISM programs are just Prolog programs that use `msw` atoms as probabilistic primitives. `msw` atoms are introduced by special declarations `values/3` specifying their properties as shown in the program DB_0 in Figure 1.

In PRISM, probabilities of ground atoms defined by a program *DB* are computed indirectly in two steps. In the first step, for a top-goal G of which we wish to compute the probability, we logically reduce it through *DB* by a top-down proof procedure, SLD search, to an equivalent propositional DNF formula $E_1 \vee \ldots \vee E_k$ such that $comp(DB) \vdash G \Leftrightarrow E_1 \vee \ldots \vee E_k$. Here each E_i $(1 \leq i \leq k)$, an *explanation for* G, corresponds to an SLD proof of G. It is a conjunction of ground `msw` atoms that records probabilistic choices made in the construction of the SLD

[3] `msw(`id`,`v`)` reads that throwing a dice named i yields an outcome v.

proof[4]. In the second step, using the fact that G and $E_1 \vee \ldots \vee E_k$ denote an identical random variable in terms of the distribution semantics for PRISM [13], we compute the probability of G as $P_{DB}(G) = P_{DB}(E_1 \vee \ldots \vee E_k)$ where $P_{DB}(\cdot)$ is the probability measure defined by *DB*.

In general there are exponentially many SLD proofs and so are explanations which result in an exponential size DNF. Nonetheless by introducing a tabling mechanism in the exhaustive SLD search process, we can often produce an equivalent but much smaller boolean formula by factoring out common sub-conjunctions in explanations as *intermediate goal*s [13, 20]. The resulting boolean formula is expressed as a conjunctive set of *defining formula*s that take the form $H \Leftrightarrow B_1 \vee \ldots \vee B_h$. Here H is the top-goal G or an intermediate goal. Hereafter the top-goal and intermediate goals are collectively called *defined goal*s. Each B_i $(1 \leq i \leq h)$ is a conjunction $C_1 \wedge \ldots \wedge C_m \wedge \mathtt{msw}_1 \wedge \ldots \wedge \mathtt{msw}_n$ $(0 \leq m, n)$ of defined goals $\{C_1, \ldots, C_m\}$ and `msw` atoms $\{\mathtt{msw}_1, \ldots, \mathtt{msw}_n\}$. We say that H is a *parent* of C_j $(1 \leq j \leq m)$. We call the closure of this parent-child relation the *ancestor relation* over ground atoms in *DB*. The whole set of defining formulas, denoted by *Exp*(G), is called the *explanation graph* for G.

2.2 From explanation graphs to probability computation

The probability $P_{DB}(G)$ of a given goal G is precisely defined in terms of the distribution semantics for PRISM. But the problem is that the semantics is so abstractly defined and we cannot know the actual value of $P_{DB}(G)$ easily. Here we describe how to compute it under some assumptions.

To compute $P_{DB}(G)$, we convert each defining formula $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ in *Exp*(G) to *a set of probability equations for* H:

$$\begin{aligned} P(H) &= P(B_1) + \cdots + P(B_h) \\ &\quad \text{where} \\ P(B_i) &= P(C_1) \cdots P(C_m) P(\mathtt{msw}_1) \cdots P(\mathtt{msw}_n) \ \ (1 \leq i \leq h) \\ &\quad \text{for } B_i = C_1 \wedge \ldots \wedge C_m \wedge \mathtt{msw}_1 \wedge \ldots \wedge \mathtt{msw}_n. \end{aligned}$$

We denote by *Eq*(G) the entire set of probability equations thus obtained. Note that the conversion assumes exclusiveness among disjuncts $\{B_1, \ldots, B_k\}$ and independence among conjuncts $\{C_1, \ldots, C_m, \mathtt{msw}_1, \ldots, \mathtt{msw}_n\}$[5]. We consider $P(H)$s in *Eq*(G) as numerical variables representing unknown probabilities and refer to them as *P-variable*s. What is important about *Eq*(G) is that *Eq*(G) *always has a solution* $P(H) = P_{DB}(H)$ *for every defined goal* H[13]. So if *Eq*(G) has a unique solution for $P(G)$, it coincides with $P_{DB}(G)$.

When defined goals appearing in *Exp*(G) are hierarchically ordered by the parent-child relation (with G as top-most element) as is usually the case, the P-variables in *Eq*(G) are also hierarchically ordered so that *Eq*(G) is uniquely and efficiently solved by dynamic programming using the generalized IO algorithm [13] in time linear in the size of *Eq*(G). There are cases however in which *Exp*(G) is not hierarchically ordered and some defined goals are their own ancestors. We say *Exp*(G) is *cyclic* if there is a defined goal having itself as an ancestor in *Exp*(G). If *Exp*(G) is cyclic, *Eq*(G) is also cyclic, and hence it is impossible to apply dynamic programming to *Eq*(G), or even worse *Eq*(G) may not have a unique solution.

---

[4] *comp*(*DB*) is the completion of *DB*. It is a union of if-and-only-if form of *DB* and so called Clark's equational theory.

[5] We assume in this paper that these conditions are always satisfied.

```
values(s,[[s,s],[a],[b]],set@[0.4,0.3,0.3]).
pre_pcfg(L):- pre_pcfg([s],L,[]).  % L is a ground list

pre_pcfg([A|R],L0,L2):-            % L0 ground, L2 variable when called
   ( values(A,_)-> msw(A,RHS),             % if A is a nonterminal
       pre_pcfg(RHS,L0,L1) ;  L0=[A|L1] ), % rule A->RHS selected
   ( L1=[] -> L2=[] ; pre_pcfg(R,L1,L2) ).
pre_pcfg([],L1,L1).
```

Figure 1 Prefix parser DB_0 for PCFGs.

In the next section, using a concrete example, we have a close look at cyclic $Eq(G)$s and investigate their properties.

3 Prefix computation for PCFGs using PRISM

In this section, we formulate prefix computation for PCFGs using PRISM.

3.1 A prefix parser

Before proceeding we introduce some terminology about CFGs for later use. Let X be a nonterminal in a CFG, α, β a mixed sequence of terminals and nonterminals. A rule for X is a production rule of the form $X \Rightarrow \alpha$. If there is a rule of the form $X \to Y\beta$, we say X and Y are in the direct left-corner relation. The *transitive closure* of the direct left-corner relation is called the *left-corner relation* and we write as $X \to_L Y$ if X and Y are in the left-corner relation. The left-corner relation is cyclic if $X \to_L X$ holds for some nonterminal X. We say that a rule is *useless* if it does not occur in any sentence derivation. A nonterminal is *useless* if every rule for it is useless. Otherwise it is *useful*. In this paper we assume that CFGs have "s" as a start symbol and have no epsilon rule and no useless nonterminal.

In addition let $\theta_1 : X \to \alpha_1, \ldots, \theta_n : X \to \alpha_n$ be the set of rules for X in a PCFG with *selection probabilities* $\theta_1, \ldots, \theta_n$ where $\sum_{i=1}^{n} \theta_i = 1$. We assume that every rule has a *positive* selection probability. If the sum of probabilities of sentences derived from the start symbol is unity, the PCFG is said to be *consistent* [18]. We also assume that PCFGs are consistent.

Now we here look at a concrete example of prefix probability computation based on cyclic explanation graphs. Consider a PCFG, $\mathbf{PG_0} = \{0.4 : \mathtt{s} \to \mathtt{s}\,\mathtt{s}, 0.3 : \mathtt{s} \to \mathtt{a}, 0.3 : \mathtt{s} \to \mathtt{b}\}$. Here "s" is a start symbol and $0.4 : \mathtt{s} \to \mathtt{s}\,\mathtt{s}$ says that the rule $\mathtt{s} \to \mathtt{s}\,\mathtt{s}$ is selected with probability 0.4 when "s" is expanded in a sentence derivation.

A PRISM program DB_0 in Figure 1 is a prefix parser for $\mathbf{PG_0}$. It is a slight modification of a standard top-down CFG parser and parses prefixes acceptable by $\mathbf{PG_0}$ such as "aab" (as list `[a,a,b]`). The difference from the usual CFG parser is that it immediately terminates successfully as soon as the input prefix is consumed even if there remain nonterminals to be processed.

`values(s,[[s,s],[a],[b]],set@[0.4,0.3,0.3])` in Figure 1 is a value declaration which encodes $\mathbf{PG_0}$. `pre_pcfg([A|R],L0,L2)` is read that `[A|R]`, a substring of α in some

```
pre_pcfg([a]) <=> pre_pcfg([s],[a],[])
pre_pcfg([s],[a],[]) <=>
   pre_pcfg([s,s],[a],[]) & msw(s,[s,s]) v pre_pcfg([a],[a],[]) & msw(s,[a])
pre_pcfg([s,s],[a],[]) <=>
   pre_pcfg([a],[a],[]) & msw(s,[a]) v pre_pcfg([s,s],[a],[]) & msw(s,[s,s])
pre_pcfg([a],[a],[])
```

Figure 2 Explanation graph for prefix "a".

rule $X \to \alpha$, spans a d-list L0-L2 as a sublist of the input list $[w_1, \ldots, w_N]$[6]. We remark that DB_0 is general, applicable to any PCFG just by replacing values/3 with appropriate value declarations.

When this program is run with PRISM-flag error_on_cycle set to "off" for a command ?-G where G = pre_pcfg($[w_1, \ldots, w_N]$) and $[w_1, \ldots, w_N]$ ($w_i \in \{\mathtt{a}, \mathtt{b}\}$) is a list representing a prefix $w_1, \ldots, w_N$, the proof procedure, the SLD search, simulates the leftmost derivation of the sentence by recursively calling the second clause. As soon as $[w_1, \ldots, w_N]$ is derived, the search terminates with success while ignoring nonterminals in R that may be non-empty as if R were successfully expanded to the remaining sentence[7]. We call this type of success *pseudo success*. During the search, a call to pre_pcfg/3 is always of the form pre_pcfg(v,$[w_i, \ldots, w_N]$,L2) where v is a substring of RHS of some production rule and $i \leq N$. On return of the call, the variable L2 is instantiated either to $[w_j, \ldots, w_N]$ ($i < j \leq N$) or to [] in the case of pseudo success. Therefore there are only a finitely many number of calling and returning patterns of prefix_pcfg/3 and hence, the tabled search for all proofs of the top-goal G always terminates.

After all proof search done, PRISM constructs an explanation graph for the top-goal G by scanning the answer table in the memory. One thing to be noticed is that goals calling themselves and thereby suspended by tabling are also recorded in the table in addition to goals that normally succeeded. When PRISM encounters such goals, it looks at the PRISM-flag error_on_cycle and if the value is "off", those goals are treated as succeeded and as a result a cyclic explanation graph is generated.

3.2 Computing prefix probabilities: an example

In this subsection, we see, using a small example, how prefix probabilities are computed from cyclic explanation graphs. Figure 2 is the explanation graph for pre_pcfg([a]) obtained by executing a command ?- probf(pre_pcfg([a]))[8] w.r.t. DB_0. As can be seen, there is a cyclic goal pre_pcfg([s,s],[a],[]) that calls itself. We convert the cyclic explanation graph to the corresponding set of probability equations shown in Figure 3. Here we used abbreviations: $\theta_{\mathtt{s}\to\mathtt{ss}} = P(\mathtt{msw}(\mathtt{s}, [\mathtt{s}, \mathtt{s}]))$ and $\theta_{\mathtt{s}\to\mathtt{a}} = P(\mathtt{msw}(\mathtt{s}, [\mathtt{a}]))$.

Although we know that the set of probability equations in Figure 3 are made true if we assign the probabilities defined by the distribution semantics[13] to X, Y, Z and W, we do not know their actual values. To know their actual values, we need to compute them by solving

[6] In the following strings beginning with lower case letters are ground terms.
[7] This is justifiable as we assume that every nonterminal is useful.
[8] probf/1 is a built-in predicate in PRISM and probf(G) displays the explanation graph of G.

$$\begin{cases} \mathtt{X} &= \mathtt{Y} \\ \mathtt{Y} &= \mathtt{Z} \cdot \theta_{\mathtt{s}\to\mathtt{ss}} + \mathtt{W} \cdot \theta_{\mathtt{s}\to\mathtt{a}} \\ \mathtt{Z} &= \mathtt{W} \cdot \theta_{\mathtt{s}\to\mathtt{a}} + \mathtt{Z} \cdot \theta_{\mathtt{s}\to\mathtt{ss}} \\ \mathtt{W} &= 1 \end{cases} \quad \text{where} \quad \begin{cases} \mathtt{X} &= P(\mathtt{pre\_pcfg}([\mathtt{a}])) \\ \mathtt{Y} &= P(\mathtt{pre\_pcfg}([\mathtt{s}],[\mathtt{a}],[])) \\ \mathtt{Z} &= P(\mathtt{pre\_pcfg}([\mathtt{s},\mathtt{s}],[\mathtt{a}],[])) \\ \mathtt{W} &= P(\mathtt{pre\_pcfg}([\mathtt{a}],[\mathtt{a}],[])) = 1 \end{cases}$$

Figure 3 Probability equations for prefix "a".

the equations. Fortunately, equations are linear in the P-variables X, Y, Z and W and easily solvable.

By substituting $\theta_{\mathtt{s}\to\mathtt{ss}} = 0.4$ and $\theta_{\mathtt{s}\to\mathtt{a}} = 0.3$[9] for the equations and solving them, we obtain $\mathtt{X} = \mathtt{Y} = 0.5$, $P(\mathtt{pre\_pcfg}([\mathtt{s},\mathtt{s}],[\mathtt{a}],[])) = \mathtt{Z} = 0.5$ and $\mathtt{W} = 1$, respectively. Hence the prefix probability of "a" is 0.5. Note that this prefix probability is larger than the probability of "a" as a sentence which is 0.3. This is because the prefix probability of "a" is the sum of the probability of sentence "a" *and* the probabilities of infinitely many sentences extending "a".

By looking at the set of probability equations in Figure 3 more closely, we can understand the way our approach computes prefix probabilities in PCFGs. For example, consider $\mathtt{Z} = P(\mathtt{pre\_pcfg}([\mathtt{s},\mathtt{s}],[\mathtt{a}],[]))$ and the equation $\mathtt{Z} = \mathtt{W} \cdot \theta_{\mathtt{s}\to\mathtt{a}} + \mathtt{Z} \cdot \theta_{\mathtt{s}\to\mathtt{ss}}$. We can expand the solution Z into an infinite series:

$$\mathtt{Z} \;=\; \frac{1}{1-\theta_{\mathtt{s}\to\mathtt{ss}}} \mathtt{W} \cdot \theta_{\mathtt{s}\to\mathtt{a}} \;=\; (1 + \theta_{\mathtt{s}\to\mathtt{ss}} + \theta^2_{\mathtt{s}\to\mathtt{ss}} + \cdots)\mathtt{W} \cdot \theta_{\mathtt{s}\to\mathtt{a}}$$

It is easy to see that this series represents the probability of infinitely many leftmost derivations of prefix "a" from nonterminals "s s" by partitioning the derivations based on the number of applications of rule $\mathtt{s} \to \mathtt{s}\,\mathtt{s}$, i.e. 1 for no application ($\mathtt{s}\,\mathtt{s} \Rightarrow_{\mathtt{s}\to\mathtt{a}} \mathtt{a}\,\mathtt{s}$), $\theta_{\mathtt{s}\to\mathtt{ss}}$ for once ($\mathtt{s}\,\mathtt{s} \Rightarrow_{\mathtt{s}\to\mathtt{ss}} \mathtt{s}\,\mathtt{s}\,\mathtt{s} \Rightarrow_{\mathtt{s}\to\mathtt{a}} \mathtt{a}\,\mathtt{s}\,\mathtt{s}$) and so on[10].

3.3 Properties of explanation graphs generated by a prefix parser

Let **PG** be a PCFG and **PG'** its backbone CFG. Also let $DB_{\mathbf{PG}}$ be a prefix parser for **PG** obtained by replacing the `values/3` declaration in DB_0 in Figure 1 with an appropriate set of `values/3` declarations encoding **PG**. In this section, we first prove that a necessary and sufficient condition under which a prefix parser $DB_{\mathbf{PG}}$ generates cyclic explanation graphs. We then prove that $DB_{\mathbf{PG}}$ always generates a system of linear equations for prefix probabilities. Finally we prove that the linear system is solvable by matrix operation under our assumptions on PCFGs.

▶ **Theorem 1.** *Let G_ℓ = `pre_pcfg(`ℓ`)` be a goal for a prefix $\ell = [w_1, \ldots, w_N]$ in* **PG'** *and $Exp(G_\ell)$ an explanation graph for G_ℓ generated by $DB_{\mathbf{PG}}$. Suppose there is no useless nonterminal in* **PG'**. *Then there exists a cyclic explanation graph $Exp(G_\ell)$ if-and-only-if the left-corner relation of* **PG'** *is cyclic.*

[9] `values(s,[[s,s],[a],[b]],set@[0.4,0.3,0.3])` in the program sets $\theta_{\mathtt{s}\to\mathtt{ss}} = P(\mathtt{msw}(\mathtt{s},[\mathtt{s},\mathtt{s}])) = 0.4$, $\theta_{\mathtt{s}\to\mathtt{a}} = P(\mathtt{msw}(\mathtt{s},[\mathtt{a}])) = 0.3$ and $\theta_{\mathtt{s}\to\mathtt{b}} = P(\mathtt{msw}(\mathtt{s},[\mathtt{b}])) = 0.3$ respectively.

[10] Recall that we assume that PCFGs are consistent. So the sum of probabilities of sentences derived from "s" is 1. Consequently for example we may ignore s in "a s" when computing the probability of prefix "a" derived from "a s".

Proof. Suppose $Exp(G_\ell)$ is cyclic. Then some defined goal `pre_pcfg([`a`|`β`],`ℓ_0`,`ℓ_2`)` with a nonterminal "a" must call itself as a descendant in $Exp(G_\ell)$ where ℓ_0 and ℓ_2 are sublists of ℓ. So an SLD derivation exists from `:-prefix_pcfg([`a`|`β`],`ℓ_0`,L2),K` to its descendant `:-prefix_pcfg([`a`|`β`],`ℓ_0`,L2'),K'` that contains no return of goals because the list ℓ_0 is preserved. Consequently there is a corresponding leftmost derivation $\mathtt{s} \stackrel{*}{\Rightarrow} a\delta \stackrel{*}{\Rightarrow} a\delta'$ by **PG'**, the backbone CFG of **PG**. So the left-corner relation is cyclic.

Conversely suppose the left-corner relation of **PG'** is cyclic. Then there is a nonterminal "a" such that $a \rightarrow_L a$. As there is no useless nonterminal by our assumption, there is a leftmost derivation starting from "`s`" such that $\mathtt{s} \stackrel{*}{\Rightarrow} \gamma a\delta \stackrel{*}{\Rightarrow} \gamma a\delta' \stackrel{*}{\Rightarrow} w_1 \ldots w_N$ for some sentence $w_1, \ldots, w_N$. In what follows, for simplicity we assume that γ is empty (but generalization is straightforward). Let $\ell_0 = w_1, \ldots, w_j$ $(j \leq N)$ be a prefix derived from a whose partial parse tree has a as the root and no a occurs below the root a. Then it is easy to see that the tabled search for all SLD proofs of G_{ℓ_0} generates $Exp(G_{\ell_0})$ containing a goal `prefix_pcfg([`a`|`β`],`ℓ_0`,[])` which is an ancestor of itself. So $Exp(G_{\ell_0})$ is cyclic. ◀

Let $Exp(G_\ell)$ be an explanation graph for G_ℓ. We introduce an equivalence relation $A \equiv B$ over defined goals appearing in $Exp(G_\ell)$: $A \equiv B$ if-and-only-if A is an ancestor of B and vice versa. We partition the set of defined goals into equivalent classes $[A]_\equiv$. Each $[A]_\equiv$ is called an *SCC* (*strongly connected component*). We say that *a defining formula* $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ *is linear* if there is no $B_i = C_1 \wedge \ldots \wedge C_m \wedge \mathtt{msw}_1 \wedge \ldots \wedge \mathtt{msw}_n$ $(1 \leq i \leq h,\ 0 \leq m, n)$ such that two defined goals, C_j and C_k $(j \neq k)$, belong to the same SCC. Also we say $Exp(G_\ell)$ *is linear* if every defining formula in $Exp(G_\ell)$ is linear.

▶ **Lemma 2.** *No two defined goals in the body of a defining formula in* $Exp(G_\ell)$ *belong to the same SCC.*

Proof. Let $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ be a defining formula in $Exp(G_\ell)$. Suppose some B_i contains two defined goals belonging to the same SCC. Looking at DB_0 in Figure 1, we know that the only possibility is such that $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ is a ground instantiation of the first (compound) clause about `pre_pcfg/3`:

```
pre_pcfg([α|β],ℓ0,ℓ2):-
    msw(a,α),pre_pcfg(α,ℓ0,ℓ1),pre_pcfg(β,ℓ1,ℓ2)                (1)
```

and the two defined goals, `pre_pcfg(`α`,`ℓ_0`,`ℓ_1`)` and `pre_pcfg(`β`,`ℓ_1`,`ℓ_2`)`, are in the same SCC. In this case, since `pre_pcfg(`α`,`ℓ_0`,`ℓ_1`)` is a proved goal, ℓ_1 is shorter than ℓ_0. On the other hand since `pre_pcfg(`β`,`ℓ_1`,`ℓ_2`)` is an ancestor of `pre_pcfg(`α`,`ℓ_0`,`ℓ_1`)` in $Exp(G_\ell)$, ℓ_0 is identical to or a part of ℓ_1, and hence ℓ_0 is equal to or shorter than ℓ_1. Contradiction. Therefore there is no such defining formula. Hence $Exp(G_\ell)$ is linear. ◀

▶ **Theorem 3.** *Let* $Exp(G_\ell)$ *be an explanation graph for a prefix* ℓ *generated by* $DB_{\mathbf{PG}}$. $Exp(G_\ell)$ *is linear.*

Proof. Immediate from Lemma 2. ◀

We next introduce a partial ordering $[A]_\equiv \succ [B]_\equiv$ over SCCs by $[A]_\equiv \succ [B]_\equiv$ if-and-only-if A is an ancestor of B but not vice versa in $Exp(G_\ell)$. We then extend this partial ordering to a total ordering $[A]_\equiv > [B]_\equiv$ over SCCs. Likewise we partition P-variables by the equivalence relation: $P(A) \equiv P(B)$ if-and-only-if $[A]_\equiv = [B]_\equiv$. We denote by $[P(A)]_\equiv$ the equivalence

class of P-variables corresponding to $[A]_{\equiv}$. By construction $[P(A)]_{\equiv}$s are totally ordered isomorphically to SCCs: $[P(A)]_{\equiv} > [P(B)]_{\equiv}$ if-and-only-if $[A]_{\equiv} > [B]_{\equiv}$. In the following we treat SCCs and P-variables as isomorphically stratified by this total ordering. We use $Eq([P(A)]_{\equiv})$ to stand for the union of sets of probability equations for defined goals in $[A]_{\equiv}$.

Notice that $Eq([P(A)]_{\equiv})$ is a system of linear equations by Theorem 3 if we consider P-variables in the lower strata as constants. Hence $Eq(G_\ell)$ is solvable inductively from lower strata to upper strata.

Now we prove that $Eq([P(A)]_{\equiv})$ is always solvable by matrix operation under our assumptions on PCFGs. Let "a" be a nonterminal in the backbone CFG **PG'** and A a defined goal in $Exp(G_\ell)$. Put $A = \texttt{pre\_pcfg([}a\texttt{|}\beta\texttt{],}\ell_0\texttt{,}\ell_2\texttt{)}$. Since A is a proved goal, A successfully calls some ground goals $B_j = \texttt{pre\_pcfg(}\alpha_j\texttt{,}\ell_{0j}\texttt{,}\ell_{1j}\texttt{)}$ shown in (1) where $a \to \alpha_j$ is a CFG rule in **PG'**. By repeating a similar proof for Lemma 2, we can prove that the third goal $\texttt{pre\_pcfg(}\beta\texttt{,}\ell_1\texttt{,}\ell_2\texttt{)}$ in the clause body in (1) does not belong to $[A]_{\equiv}$, the SCC containing A. Thus $[A]_{\equiv} > [\texttt{pre\_pcfg(}\beta\texttt{,}\ell_1\texttt{,}\ell_2\texttt{)}]_{\equiv}$. So only some of the B_js can possibly belong to $[A]_{\equiv}$ as far as A is concerned.

Let $P(A_1), \ldots, P(A_K)$ be an enumeration of P-variables in $[P(A)]_{\equiv}$. Introduce a column vector $X_A = (P(A_1), \ldots, P(A_K))^T$. It follows from what we have argued that we can write $Eq([P(A)]_{\equiv})$ as a system of linear equations $X_A = MX_A + Y_A$ where M is a $K \times K$ non-negative matrix and Y_A is a non-negative vector whose component is a sum of P-variables in the lower strata multiplied by constants. M is irreducible because in $Exp(G_\ell)$, every goal in $[A]_{\equiv}$ directly or indirectly calls every goal in $[A]_{\equiv}$. Y_A is non-zero because some A_i must have a proof tree that only contains defined goals in the lower strata. For vectors U, V, we write $U > 0$ (resp. $U \geq 0$) if every component of U is positive (resp. non-negative) and $U \geq V$ if $U - V \geq 0$ where 0 is a zero vector.

▶ **Theorem 4.** *Let* **PG** *be a consistent PCFG such that there is no epsilon rule and every production rule has a positive selection probability. Also let* $DB_{\mathbf{PG}}$ *be a prefix parser for* **PG** *and* $Exp(G_\ell)$ *an explanation graph for a prefix* ℓ*. Suppose* $Eq([P(A)]_{\equiv})$ *is a system of linear equations for a defined goal* A *in* $Exp(G_\ell)$*. Put* $[P(A)]_{\equiv} = \{P(A_i) \mid 1 \leq i \leq K\}$ *and write* $Eq([P(A)]_{\equiv})$ *as* $X_A = MX_A + Y_A$ *where* $X_A = (P(A_1), \ldots, P(A_K))^T$*. It has a unique solution* $X_A = (I - M)^{-1}Y_A$*.*

Proof. We prove that $I - M$ has an inverse matrix. To prove it, we assume hereafter that P-variables in $[P(A)]_{\equiv}$ are assigned as their values probabilities defined by the distribution semantics and hence all equations in $Eq([P(A)]_{\equiv})$ are true.

By applying $X_A = MX_A + Y_A$ k repeatedly to itself, we have $X_A = M^k X_A + (M^{k-1} + \cdots + I)Y_A$ for $k = 1, 2, \ldots$ Since M, X_A, and Y_A are non-negative, we have $X_A \geq M^k X_A$ and $X_A \geq (M^{k-1} + \cdots + I)Y_A$ for every k. On the other hand since $\{(M^{k-1} + \cdots + I)Y_A\}_k$ is a monotonically increasing sequence of non-negative vectors bounded by X_A, it converges and so does $\{M^k X_A\}_k$.

Let $\rho(M)$ be the spectral radius of M[11]. Suppose $\rho(M) > 1$. In general $\rho(M) \leq \| M^k \|_\infty^{\frac{1}{k}}$ holds for every k where $\| \cdot \|_\infty$ is the matrix norm induced from the ∞ vector norm. It follows from $\rho(M)^k \leq \| M^k \|_\infty$ that $\lim_{k\to\infty} \| M^k \|_\infty = +\infty$. Consequently since $X_A > 0$ holds

[11] $\rho(M)$ is the largest eigenvalue of M. As M is irreducible, the right eigen vector and the left eigen vector associated with $\rho(M)$ are both positive by the Perron-Frobenius theorem.

because every proved goal has a positive probability from our assumption, some element of $M^k X_A$ goes to $+\infty$, which contradicts the convergence of $\{M^k X_A\}_k$. So $\rho(M) \leq 1$.

Suppose now $\rho(M) = 1$. Then in this case, we note that $\left\{\frac{M^{k-1}+\cdots+I}{k}\right\}_k$ converges to a positive matrix (proof omitted), and hence $(M^{k-1}+\cdots+I)Y_A = \left(\frac{M^{k-1}+\cdots+I}{k}\right)\cdot kY_A$ diverges as k goes to infinity, which contradicts again the convergence of $\{(M^{k-1}+\cdots+I)Y_A\}_k$. Therefore $\rho(M) < 1$. So $(I-M)^{-1}$ exists. ◀

Note that $X_A = (I-M)^{-1}Y_A = (I+M+M^2+\cdots)Y_A$. By further analyzing the matrix M, we understand that multiplying M by Y_A for example corresponds to growing partial parse trees by one step application of production rules (reduce operation in bottom-up parsing). Hence $P(A_i)$, a component of X_A, is an infinite sum of probabilities and so is the probability of the top prefix goal $P(\texttt{pre\_pcfg}(\ell))$.

Summing up, we compute prefix probabilities for a PCFG **PG** as follows. Let *DB* be a prefix parser for **PG** and $G = \texttt{pre\_pcfg}(\ell)$ a goal for a prefix ℓ.

[Step 1]: From G and *DB*, construct an explanation graph $Exp(G)$.
[Step 2]: Extract the set of probability equations $Eq(G)$ from $Exp(G)$.
[Step 3]: Solve $Eq(G)$ inductively from lower strata by matrix operation and obtain $P_{DB}(G)$, the prefix probability of ℓ.

The above procedure is general and applicable to arbitrary (cyclic) linear explanation graphs, not restricted to those generated by a PCFG prefix parser. We computed prefix probabilities for PLCGs (probabilistic left-corner grammars) similarly to PCFGs, but we omit the detail due to space limitations.

4 Related work

Prefix probability computation is mostly studied about PCFGs [3, 15, 7]. Jelinek and Lafferty [3] proposed a CKY like algorithm for prefix probability computation in PCFGs in CNF (Chomsky normal form). Their algorithm does not perform parsing but instead uses a single matrix whose dimension is the number of nonterminals which is constructed from a given PCFG. It runs in $O(N^3)$ where N is the length of an input prefix. Stolcke [15] applied the Earley style parsing to compute prefix probabilities. His algorithm uses a matrix of "probabilistic reflexive, transitive left-corner relation" computed from a given PCFG, independently of input sentences similarly to [3]. Our approach differs from them in that it works for probabilistic logic programs and it deals with explanation graphs constructed for each input prefix. Nederhof and Satta [7] generalized prefix probability computation for PCFGs to infix probability computation for PCFGs. They also studied prefix probability computation for a variant of PCFGs [8]. Nederhof et al. proposed prefix probability computation for stochastic tree adjoining grammars [6].

Approximate computation of prefix probabilities is possible for example by the iterative deepening algorithm used in ProbLog[2], but it is out of the scope of this paper.

5 Conclusion

We have proposed an innovative use of tabling: infinite probability computation based on cyclic explanation graphs generated by tabled search in PRISM. Our approach generalizes

prefix probability computation in PCFGs and is applicable to probabilistic models described by PRISM programs in general as well as PCFGs. In particular it is applicable to non-PCFG probabilistic grammars such as PLCGs though we omitted the result of prefix computation for PLCGs due to space limitations. We are developing a tool that generates a (cyclic) explanation graph for a given goal and computes its probability by solving the system linear equations associated with it. We expect that our approach provides a declarative way of logic-based probabilistic modeling of cyclic dependencies.

References

1 J. K. Baker. Trainable grammars for speech recognition. In *Proceedings of Spring Conference of the Acoustical Society of America*, pages 547–550, 1979.

2 L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2468–2473, 2007.

3 F. Jelinek and J. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323, 1991.

4 C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

5 T. Mantadelis and G. Janssens. Dedicated tabling for a probabilistic setting. In *Proceedings of the 26th International Conference on Logic Programming (ICLP'10) (Technical Communications)*, pages 124–133, 2010.

6 M. Nederhof, A. Anoop Sarkar, and G. Satta. Prefix probabilities from stochastic tree adjoining grammars. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL'98)*, pages 953–959, 1998.

7 M. Nederhof and G. Satta. Computation of infix probabilities for probabilistic context-free grammars. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP'11)*, pages 1213–1221, 2011.

8 M. Nederhof and G. Satta. Prefix probability for probabilistic synchronous context-free grammars. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'11)*, pages 460–469, 2011.

9 F. Riguzzi and T. Terrance Swift. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming (TPLP)*, 11(4-5):433–449, 2011.

10 R. Rocha, F.M.A. Silva, and V.S. Costa. On applying or-parallelism and tabling to logic programs. *Theory and Practice of Logic Programming (TPLP)*, 5(1-2):161–205, 2005.

11 T. Sato. A glimpse of symbolic-statistical modeling by PRISM. *Journal of Intelligent Information Systems*, 31(2):161–176, 2008.

12 T. Sato and Y. Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1330–1335, 1997.

13 T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

14 T. Sato and Y. Kameya. New Advances in Logid-Based Probabilistic Modeling by PRISM. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic Inductive Logic Programming*, pages 118–155. LNAI 4911, Springer, 2008.

15 A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

16 H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proceedings of the 3rd International Conference on Logic Programming (ICLP'86)*, volume 225 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1986.

17 D. S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):93–111, 1992.

18 C. S. Wetherell. Probabilistic languages: a review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

19 N.-F. Zhou, Y. Kameya, and T. Sato. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proceedings of the 22th International Conference on Tools with Artificial Intelligence (ICTAI-2010)*, 2010.

20 N.-F. Zhou, T. Sato, and Y.-D. Shen. Linear tabling strategies and optimization. *Theory and Practice of Logic Programming (TPLP)*, 8(1):81–109, 2008.